# דפוסי עיצוב - Design Patterns

דפוסי עיצוב בהם ייעשה שימוש בפרויקט:

## MVVM – Model-View-ViewModel

תבנית זו תפקידה לייצר הפרדה בין המסכים (הViews) לבין אובייקטי המידע (Models) והלוגיקה של המסכים (הViewModels). כך ניתן להחליף כל אחד מרכיבים אלה עם מעט מעוד שינויים בשכבות האחרות (בדרך כלל אם אין שבירת ממשקים בין השכבות אז גם אין שינויים בכלל).

בתבנית זו נעשה שימוש בPresentation Layer כך שלכל מסך יש את הView Model והViewModel שלו. ניתן לראות את ה"חיבור" בין הViews לViewModels בקובץ App.xaml על ידי DataTemplates, מה שגורם לכך שבכל פעם שנרצה להציג על המסך ViewModel מסוים, יוצג באופן אוטומטי הView הרלוונטי שלו.

## Singleton

תבנית שתפקידה לוודא שקיים למחלקה מופע אחד בדיוק, בנוסף התבנית מספקת מצביע גלובלי שיאפשר לגשת למופע זה. שימוש בתבנית זו ימנע מהיווצרותן של שגיאות לוגיות הנגרמות כתוצאה מיצירה של כמה מופעים עבור מחלקות שנדרש עבורן מופע אחד לכל היותר.

בתבנית זו נעשה שימוש במחלקה AuthenticationTokenSingleton.

מחלקה זו מייצגת את הAuthenticationToken של המשתמש הנוכחי שמחובר למערכת. מאחר שבכל רגע נתון בכל עמדת קצה רק משתמש אחד יכול להיות מחובר אל המערכת, נעשה שימוש בתבנית Singleton במחלקה זו. שימוש בתבנית יעזור לנו לוודא שלא קיים יותר ממשתמש אחד שמחובר למערכת – כלומר שיש רק AuthenticationToken אחד (שהשימוש שלו זה הזדהות מול שכבת הBL). הAuthenticationTokenSingleton נגיש לכל מחלקה בPresentation Layer, וניתן באמצעותו גם להשיג מידע על המשתמש המחובר מהBL (למשל לבדיקת הרשאות).

## Observer

תבנית המגדירה תלות אחד-לרבים בין האובייקטים, כאשר מצב של אובייקט אחד משתנה, כל שאר האובייקטים שתלויים בו מקבלים התראה על השינוי ומתעדכנים באופן אוטומטי.

בתבנית זו נעשה שימוש במחלקות הבאות:

DiscussionThreadObserver –

מחלקה זו מייצגת ומודיעה על הגרסה העדכנית ביותר של הDiscussion Thread שנצפה כעת (במידה והמסך הנוכחי הוא מסך צפייה בשאלה והתשובות שלה. מחלקה זו בעצם תודיע לכל מחלקה שמבקשת, בכל פעם שנעשה שינוי והתעדכן משהו בDiscussion Thread הנוכחי. היא תעשה זאת על ידי כך שכל כמה שניות היא פונה לשכבת הBL ובודקת האם שדה הVersion של אותו Discussion Thread בו אנחנו צופים כעת, גבוהה יותר מהVersion הנוכחי שנצפה כעת. כאשר הDiscussionThreadObserver מזהה שהגרסה גדלה, הוא מודיע בEvent לSubscriber שלו – שבמקרה זה זו המחלקה QuestionViewModel – שעליו לעדכן את הDiscussion Thread אותו הוא מציג לאחד חדש יותר, ודרך הEventArgs מעביר לו את הDiscussionThread החדש אותו עליו להציג.

NotificationsObserver –

מחלקה זו מודיעה לכל דורש על כל ההתרעות החדשות שרלוונטיות למשתמש המחובר.

בכל כמה שניות הNotificationsObserver פונה לשכבת הBL ומבקש התרעות חדשות עבור המשתמש המחובר. בתשובה הוא יקבל רק התראות חדשות שלא קיבל כבר בעבר מהBL, ויודיע על ההתרעות החדשות על ידי Event. הSubscriber היחיד כרגע של הNotificationsObserver הוא הNotificationsViewModel שמציג את כל ההתראות שרלוונטיות למשתמש. ההתרעות החדשות יועברו על בEventArgs של הEvent.

## Command

תבנית שעושה אנקפסולציה לבקשה בתור אובייקט, כך שניתן להעביר את אובייקט זה בין מחלקות כאובייקט, וכל מחלקה יכולה להפעיל את הבקשה עם פרמטרים משלה.

תבנית זו ממומשת על ידי המחלקה DelegateCommand שבה משתמשים בכל שכבת הPL. השימוש בתבנית יעשה כך, שהViewModel יוכל להגדיר Commandים שונים לפעולות שונות, ולחשוף אותם לViews דרך Binding כך שהView יוכל להפעיל את הCommandים עם פרמטרים שלו (שקיבל למשל דרך User Input). כמו כן DelegateCommand מממש גם CanExecute, כך שהView יכול להציג למשתמש האם הוא יכול לבצע את הבקשה או שלא (למשל על ידי העברת כפתור למצב Disabled).

## Decorator

תבנית שמאפשרת לחבר מספר אחריויות לאובייקט דינמי אחד, כך שהפעלה של אובייקט זה יפעיל את כל האחריויות שמחוברות לאובייקט. בצורה זו, השימוש באובייקט יכול להיות באין יודעין על כל המחלקות ש"עוטפות" את האובייקט הנתון.

השימוש בתבנית זו נעשה במחלקה ValidatorDecoratorובכל המחלקות היורשות ממנה:

ContentValidator, MarkRightAnswerValidator, TitleValidator, EmailAddressValidator, NameValidator, PasswordValidator, RankDownValidator, RecommendQuestionValidator.

כל מחלקה כזאת יורשת מValidatorDecorator וכך ניתן לכל פעולה (למשל רישום של משתמש חדש) ליצור "שרשרת" של בדיקות שונות כך שכל Validator עוטף Validator אחר, ולבסוף מתקבל אובייקט Validator יחיד שמבצע את כל הבדיקות הדרושות על מנת לבדוק האם הרישום של המשתמש למערכת תקין. בדרך זו ניתן לעשות Reuse לוולידציה כך שניתן יהיה להשתמש בה במקומות שונים במערכת (למשל גם בPL וגם בBL) .שרשרת הValidations נבנית פעם אחת בכל מחלקה, ואז כל השיטות של המחלקה משתמשות בוולידטור בלי ידיעה מאיזה וולידטורים הוא בנוי. אם ברצוננו להוסיף וולידציה חדשה לפעולה מסוימת (למשל לרישום משתמש חדשה) עלינו רק להוסיף בבניית השרשרת את המחלקה החדשה היורשת גם היא מValidatorDecorator, ולא לשנות את השיטות עצמן שקוראות לValidatorDecorator ומבקשות ממנו לבצע וולידציה.

# שינויים עתידיים אפשריים

המערכת תוכננה לעוצבה כך ששינויים עתידיים יהיו קלים ליישום. נראה כעת כיצד הגמישות לשינויים באה לידי ביטוי ואילו שינויים ניתן יהיה לבצע בעתיד.

1. המערכת בנויה על סמך מודל שלושת השכבות 3-Tier. הפרדה זו יוצרת שכבות אבסטרקציה, כך שבעתיד ניתן יהיה להחליף כל אחת מהשכבות בלי להחליף את שאר השכבות. או להחליף ממשק בין שתי שכבות, בלי להחליף את הממשק בין שתי שכבות אחרות. לדוגמה, אם נחליט שברצוננו לעבוד עם מסד נתונים שונה, או עם framework שונה כמו NHibernate נוכל לעשות זאת ללא כל שינוי בשכבות הBL ובPL. דוגמה נוספת לשינוי היא למשל החלפת שכבת הPL במימוש WEBי (כמעט) ללא שום שינוי בשכבת הBL\DAL (השינוי היחיד שיש לבצע הוא בהגדרת הServices של הBL כך שניתן יהיה לפנות אליהם באמצעות Rest API ושינוי זה הוא מאוד מאוד קטן).
2. הוספת מסכים חדשים בשכבת התצוגה תתבצע בקלות רבה. חוץ מכתיבת המסך החדש, כל שעלינו לעשות הוא להוסיף את הקוד שמבצע מעבר למסך החדש – כלומר למשל אם נוסיף כפתור חדר לMainMenuViewModel, כל שנצטרך לעשות הוא להוסיף לו Command שיוצר מופע חדש של המסך החדש וקורא לNavigationManager כדי שיעבור למסך החדש.
3. שינוי מסך קיים – מאחר שמימשנו את שכבת הPL בעזרת MVVM, ניתן יהיה בקלות רבה לשנות את איך שכל מסך נראה על ידי שינוי הView בלבד, ללא שום שינוי בViewModel. גם ההפך נכון, ניתן לשנות למשל מה כפתור מסוים מבצע או פעולת משתמש מסוימת מבצעת רק על ידי שינוי בViewModel ללא שום שינוי בView. למשל אם נרצה למשל לשנות את הכפתורים בMainMenu לHyperlink, נוכל לעשות זאת בקלות בView מבלי לשנות את הקוד שמבצע את הNavigation בViewModel.
4. הוספת התרעות חדשות – הצורה שבה בנויה תשתית ההתרעות מאפשרת בעתיד להוסיף התרעות חדשות בקלות. בDB נשמרות ההתראות בטבלה בה רשומה ההתראה, והמשתמש שמיועד לקבל את ההתרעה. בDAL תהיה פעולה שמאפשרת לBL לפנות אליה ולהוסיף את ההתרעה החדשה לDB. בדרך זו, בכל פעולה ניתן להוסיף לBL פניה לDAL שישמור התרעה חדשה עבור מבצע הפעולה (ואולי עוד משתמשים – תלוי בפעולה). למשל, אם נחליט שהזנת שאלה למערכת תודיע לרכז, אז בPostServices בפועלה של הוספת שאלה חדשה, נוסיף קריאה לDAL של שמירת ההתרעה עבור הרכז.
5. הוספת וולידציות חדשות או שינוי קיימות – מאחר שמימשנו את מנגנון הוולידציות במערכת באמצעות Decorator, כל וולידציה יושבת במחלקה נפרדת. כך, אם נרצה לשנות את הלוגיקה בבדיקה מסוימת, נוכל לשנות אך ורק את המחלקה הספציפית שמבצעת את הבדיקה שברצוננו לשנות, ללא כל שינוי של מחלקות אחרות. בנוסף, אם נרצה להוסיף וולידציה חדשה לפעולה כלשהי במערכת, יהיה ניתן או לעשות Re-Use לוולידציה קיימת, או ליצור מחלקה חדשה של וולידציה, ולהוסיף אותה כחלק מהבניה של הDecorator עבור אותה פעולה. הכוונה בבנייה של Decorator היא החלק בו אנחנו בונים decorator על decorator עד אשר נשאר מופע בודד של validator.
6. שינוי תוכן הAuthenticationToken – העיצוב הנוכחי של הפרויקט, כתוב כך שהPL אינו מודע אף פעם לתוכן של הAuthenticationToken שהוא מקבל מהBL לאחר חיבור מוצלח למערכת. הדבר היחיד שהPL עושה הוא לשמור את הAuthenticationToken אצלו ולהעביר אותו לBL על מנת לזהות את המשתמש שמבקש לבצע פעולה. בדרך זו הBL הוא היחיד שיודע לכתוב ולפענח את הAuthenticationToken. בדרך זו ניתן למשל בעתיד להחליט למשל להצפין את התוכן של הAuthenticationToken בכל פעם שמשתמש מנסה להתחבר למערכת. ההצפנה יכולה להיות תלויה למשל בתאריך והשעה הנוכחיים כך שבמידה ותהיה מתקפה כלשהי על המערכת על ידי פורצים, הם לא יוכלו לדעת להפיק AuthenticationToken משלהם כדי לבצע פעולות בשם משתמשים קיימים במערכת.